UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/650,238	08/27/2003	Wolfram Schulte	3382-65592	6392

26119 7590 06/09/2009
KLARQUIST SPARKMAN LLP
121 S.W. SALMON STREET
SUITE 1600
PORTLAND, OR 97204

EXAMINER	
VU, TUAN A	

ART UNIT	PAPER NUMBER
2193	

MAIL DATE	DELIVERY MODE
06/09/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	**Application No.**	**Applicant(s)**	
	10/650,238	SCHULTE ET AL.	
	Examiner	**Art Unit**	
	TUAN A. VU	2193	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *27 May 2009*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,3-5 and 7-33* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,3-5 and 7-33* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.	6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the Applicant's response filed 5/27/09.

As indicated in Applicant's response, claims 1 have been amended. Claims 1, 3-5, 7-33 are pending in the office action.

Claim Objections

2. Claims 1, 19, 33 are objected to because of the following informalities: the phraseology written as 'target efficient testing of the program when executed' (cl. 1, 19, 33) and 'targeting testing' (cl. 1) amounts to idiomatic English constructs that cannot be unduly understood. The disclosure does not provide a single passage explaining what 'targeting testing' actually means, nor does one acknowledge how 'target efficient testing of ... program when executed' is implemented therein in terms of the 'target efficient testing ... when executed' exact syntax. The above obscure syntaxes will be treated as mere code testing a target program. Appropriate correction is required.

Claim Rejections - 35 USC § 112

3. The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

4. Claims 1, 3-5, 7-18 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention.

Specifically, the language recited as '<u>reading a reflection</u> of the executable … program <u>during a first execution</u> of the program' (cl. 1) is not shown to have proper support or clear description in the Specifications. Scanning the term 'reflection', it is observed that 'reflection' is but merely metadata regarding types or data structures used in a program (Specifications: page. 10, and Fig. 5) such that *the program* in that context is not an executable being run (as recited in the claim) in the course of which one first runtime metadata is collected. The configuration stage by which the domain manager 410 (Fig. 5) utilizes configuration information, reflection and annotates domain data structure cannot be accepted as a runtime of an executable code within which reflection data is dynamically retrieved. Specifically, this disclosed *configuration manager 410* <u>may</u> be able to collect data structure information about types, fields, methods by reading a reflection, in conjunction with receiving domain configuration 430 (i.e. "receiving domain configuration information") and produces domain data 440 using the reflection information and configuration 430 (Specs, pg. 10). The reading done by *manager 410* is after the fact that reflection has been made available, no reading done while code (of first execution) is running. The language regarding 'during a first execution of the program' is not deemed a feature the Inventor possesses at the time the Invention was made. This 'first execution' for enabling a concurrent reception of 'reflection' data will not be given any patentable weight; and will be treated as a mere framework runtime where manager code operates to collect program-related data types or metadata data structure.

Claim 1 recite 'targeting testing … **during** a second execution of the computer program' (cl. 1 line12)

From scanning the Specifications, the term ‘executing’ is encountered once when an example is given about introspection capability known in .NET executing program or Sun’s Java language (bottom pg. 10). The claim does not describe a *first* execution using those introspection methods to capture runtime reflection, making the ‘reading a reflection’ step clearly subsequent to any execution for invoking this reflection, hence no 'during a first execution' being enabled by the context wherein *domain manager 410* reads a reflection (see above). Nor does the Disclosure mention about any ‘runtime’ or ‘execution' or some instance(s) of ‘executing’ a **same** program any time after reflection has been read by manager 410 (Note: Figure 5 only teaches compiling annotated data into a domain data – step 540). The *first* execution of a program followed by a *second* execution of that same program is not deemed as having explicit and reasonable support.

However, based on the desirability (see *It may be desirable* - Specs: pg. 8) to target a verification of software by selecting or limiting a domain of values used in the testing the ‘first execution’(claim 1) will be treated as a software pre-translation stage in which *reflection data* and *configuration information* are processed together towards controlling or verifying proper execution of a program, whereas the ‘during a/the second execution’ will be viewed as a test stage which includes ‘determining whether the executable program behaves correctly when … falling within data domain’; that is, targeting testing given weight to the extent of the ‘determining whether …’ limitation, and ‘during first execution’ not given weight.

Dependent claims 3-5, 7-18 are also rejected for the lack of enablement in view of the above.

Claim Rejections - 35 USC § 103

5. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

6. Claims 1, 3-5, 7-33 are rejected under 35 U.S.C. 103(a) as being unpatentable over Davidson et al., USPN: 6,083,276(hereinafter Davidson), and further in view of Java2SE, "Class Introspector", Java 2 Platform, Sun Microsystems ed. ver. 1.4.0, Copyright 1993-2002 pp. 1-6(hereinafter Java2SE).

As per claim 1, Davidson discloses a computer implemented method for producing a data domain for a data structure element of an executable computer program to be used to target efficient testing of behavior of the program when executed, the executable computer program having been compiled into executable form, the method comprising:

receiving domain configuration information (e.g. Fig. 3A, 3B, 3C) corresponding to the data structure element;

reading a reflection of the executable computer program bean a property (e.g. Table 1; *BeanInfo* -Fig. 5; *standard reflection functionality built into Java … runtime inspection* – col. 26, lines 50-60; *accessor ... methods* … value of a property – col. 25, lines 1-8 – Note: beanInfo reads on a form of reflection information on bean) during a first execution of the program (refer to USC 112 Rejection);

producing the data domain based on the domain configuration information and the reflection of the executable computer program(Fig. 4B, 4C – Note: mapping corresponding descriptor or attribute for a method or class reads on data domain combining configuration information with reflection of beans components – see col. 25, line 12 to col 26, line 9),

the data domain representing a limited set of data values to be used as input (e.g. *expected parameters* – col. 25, lines 8-32; *canonical names ... scope path named ... more attributes ... remain to be mapped* – col. 25, line 34 to col. 26, line 9 – Note: verifying correctness of parameters, basic types, canonical objects in light of their expected number or instances being enclosed within some Descriptor scope reads on testing computer program so that data structure domain values fall under that structure – see *propertyDescriptor, parameters ... write method* - Fig. 4C) during a subsequent second execution of the computer program (refer to USC 112 second paragraph – second execution treated as 'determining whether ... behaves correctly');

targeting testing *during the second execution* (refer to USC 112 second paragraph – 'during the second execution' subsumed in the 'determining whether' step) to use only values for the data structure element that fall within the data domain and determining whether the executable computer program behaves correctly (e.g. generate error 458 - Fig 4D – Note: error generated reads on dynamic execution and determining; wherein verifying correctness of parameters, basic types, canonical objects in light of their expected number or instances being enclosed within some Descriptor scope reads on testing computer program so that data structure domain values fall under that structure or domain info of the bean container or Property scope – e.g. <Style type=Link … value= … value=… … value= … </Style> , pg. 21 lines 40-48; col. 26, lines 18-28; or enumerating values in a array – see class[] and Object[] lines 38-48) during the second execution.

Davidson does not explicitly disclose targeting testing as executing *executable computer program* when executed using targeted values falling within the data domain as input. Davidson discloses prospect of type violation in a execution environment (generate error 458 - Fig 4D – Note: error generated via verification reads on potential violation in dynamic execution), a form of verification to determine whether some expected value or range of Java elements (composing a container scope for a specific method or bean container) when used in a runtime would be proper, such runtime conflict-prevention approach being based on the domain data provided by the results from parsing XML into a structure (e.g. Fig. 3B), with the type of metadata (e.g. Fig. 4C) being received from using bean info descriptor and accessor API (e.g. *executable 'accessor' methods ... obtaining and setting the value of a property 32*0 – col. 24, line 50 to col. 25, line 7) or alternatively, using API to introspect the runtime bean (e.g. *beanInfo instance 124, introspection* -col. 26, lines 55-60; beaninfo – Fig. 5). Hence, the prospect of having code tested and executing this code for evaluating its proper behaviour is strongly suggested. Further, standardardized APIs thus mentioned, being invoked when executing Sun Microsystems Java or bean code to dynamically obtain additional information about code strongly entails dynamic type of data in order to help enhance the code or prevent its potential undesired use of runtime resources similar to debug or validation testing. Accordingly, Java2SE discloses APIs to support introspection on target bean (Method getBeanInfo – pg. 3; getBeanInfoSearchPath – pg. 4) Hence, the endeavor of dynamic verification support when running a target program is recognized from above; e.g. Java runtime of a bean being captured to support debug of bean behavior as by Java2SE; runtime exception observed from code verificationfrom Davidson' endeavor for dynamic support for running target applications – see Davidson: col. 4 li. 33-45;

runtime binding – col 6 lines 16-26), such that runtime of target program would be based on dynamic support from meta-information (*beanInfo* via the well-known reflection/introspection APIs) received, for example, from a bean runtime introspector API in terms that *standard reflection methods* (see Java2SE) as this alternative is strongly evidenced in Davidson from above.

It would have been obvious for one skill in the art at the time the invention was made to implement Sun Microsystems API in Davidson's verification/test of a target bean code as shown above, so that reflection data can be obtained using such runtime API (Davidson: *standard reflection functionality built into Java … runtime inspection* – col. 26, lines 50-60) and using such collected information to implement verification of target program in terms of running a compiled target code (executable bean) based on the collected reflection data (e.g. *beanInfo* via introspection) serving as determining domain input; i.e. verifying whether the executing bean in light of such reflection data behaves according to some expected value or range of Java elements composing a container scope for a specific method or bean container.

As per claim 3, refer to claim 1 for a listing of data structure elements of the computer program as reflection of computer program (e.g. Fig. 2).

As per claim 4, Davidson discloses annotating code of the computer program (e.g. comment 302 – Figs. 3; e.g. col. 29-36 Appendix A for ADML for comments between special tags <!-- ... -->) with the domain configuration information.

As per claim 5, Davidson discloses computer readable media having computer executable instructions for compiling the code of the computer program annotated with the domain configuration information for producing the data domain (Fig. 4B, 4C cols 21-28) according to its domain configuration information.

As per claims 7-8, Davidson discloses the domain configuration information comprising one or more expressions (e.g. BML – col. 8-col. 10 – Note: tag specification *<Foo Att1 = Value 1 Att2=Value2 ... />* reads on explicit denotation of domain to be produced) for explicitly denoting the data domain to be produced corresponding in form to one that is applicable to the data structure element; wherein the expressions comprise methods and functions (e.g. parameter method Fig. 4C; METHOD, ARGUMENTS - col. 16, lines 28-42; Fig. 5; *CALL calls a method ... Attributes* - Appendix A, col. 47, bottom - Note: beans constructs being described in BML language as method and arguments reads on methods and functions) defined within the code of the computer program, which are exposed via the reflection of the computer program.

As per claim 9, Davidson discloses wherein the data structure element is a data type with one or more fields and the form of the explicitly expressed data domain is a set of values of the fields comprising the data type (e.g. TYPE – col. 16, lines 54-63).

As per claim 10, Davidson discloses wherein the data structure element is a method (re claim 9) and the form of the explicitly expressed data domain is a set of tuples of parameters (e.g. *GET FIND CONSTANT|ARRAY| ANY| ALL|NOT* – Appendix A, col. 29-30) of the method.

As per claim 11, Davidson discloses wherein the data structure element is a field or a parameter of a designated type (Table 2, pg. 19; TYPE ID - lines 55-63, col. 16) and the form of the explicitly expressed data domain is an enumeration of values (e.g. <VALUE ...</VALUE> lines 55-63, col. 16; lines 28-42, col. 16) of the designated type corresponding to the field or the parameter.

As per claim 12, Davidson discloses inheriting (e.g. Fig. 3C; *children* - col. 10, lines 50-65; *Child component ...Parent component* – Fig. 4D; lines 9-29 - col. 13) the data domain to be produced from the data domain of other related data structure elements.

As per claim 13, Davidson discloses a data type comprising a plurality of sub-types and a selection of one or more of the plurality of sub-types wherein the data domain to be produced for the data type is a union of data domains of the sub-types (P tag ... Table 2, col. 19; Style tag...Table 3, col. 20 – Note: paragraph and style tag with subtypes read on plurality of subtype and selection from an union of subtypes) belonging to the selection.

As per claim 14, Davidson discloses data structure element being a field or a parameter of a designated type (Fig. 5; Table 2, pg. 19; TYPE ID - lines 55-63, col. 16; cols. 17-18) and the domain configuration information comprises information indicating that the data domain to be produced for the field or the parameter is inherited (e.g. Fig. 3C; *children* - col. 10, lines 50-65; *Child component... Parent component* – Fig. 4D; lines. 9-29 - col. 13)from the data domain of their designated type.

As per claims 15-16, Davidson discloses domain configuration information related to producing the data domain for the data structure element by applying domain generation techniques on other selected data domains; and filtering the result of the applying domain generation technique step using a predicate (steps 410, 418, Fig. 4B; Match 424, Conversion 432, More attributes 440 – Fig. 4C; step 456, Fig. 4D – Note: code to map components as called by description information with respect to parse algorithm reads on predicates for filtering data from information domain into data domain – see Java pseudo-code col. 26, 28).

As per claims 17-18, Davidson discloses data structure element is a data type with a plurality of fields (e.g. lines 30-34, 55-58 -col. 9; lines 30-43, col. 16; Table 2, col. 19) and the other data domains are data domains of the fields (re claim 1 or Fig. 4B, 4C); wherein the data structure element is a method (Table 1, col. 17-18) and the other data domains are data domains of the parameter of the method (re claim 1; see *write method 504* - col. 25).

As per claim 19, Davidson discloses a system for producing a data domain for a data structure element of an executable computer program to be used to target efficient testing of the program when executed, the executable computer program having been compiled into executable form, the system comprising a computer apparatus configured to perform the actions of a domain configuration manager for:

receiving domain configuration information corresponding to the data structure element (refer to claim 1);

using the reflection of the executable computer program (e.g. bean a property Table 1; *BeanInfo* -Fig. 5; *standard reflection functionality built into Java … runtime inspection* – col. 26, lines 50-60; *accessor ... methods …* value of a property – col. 25, lines 1-8 – Note: beanInfo reads on a form of *reflection* information on bean) to produce the data domain for the data structure element according to the domain configuration information (Fig. 4B, 4C – Note: mapping corresponding descriptor or attribute for a method or class reads on data domain combining configuration information with reflection of beans components – see col. 25, line 12 to col 26, line 9),

the data domain representing a limited set of data values to be used as input for testing the executable computer program when executed (e.g. col. 26, lines 18-29, 38-48; *setting the* *value of a property 320* – col. 24, line 50 to col. 25, line 7) to be used as input for testing the executable computer program when executed (Execute 436 – Fig. 4C – Note: exception error reads on validating or verifying on a range of value or scope of expected constructs being used as input; and whereby error is generated – as a result of on mismatch of expected parameter or method attributes against domain information **reads on** during execution of program to be tested - e.g. <Style type=Link … value= … value=… … value= … </Style> , pg. 21 lines 40-48; col. 26, lines 38-48; enumerating values in a array – see class[] and Object[] lines 38-48); and

controlling testing to use only values for the data structure element that fall within the data domain (e.g. *expected parameters* – col. 25, lines 8-32; *canonical names … scope path named … more attributes … remain to be mapped* – col. 25, line 34 to col. 26, line 9 – Note: verifying correctness of parameters, basic types, canonical objects in light of their expected number or instances being enclosed within some Descriptor scope reads on testing computer program so that data structure domain values fall under that structure – see *propertyDescriptor, parameters … write method* - Fig. 4C).

Davidson does not explicitly disclose *producing a reflection* of the executable computer program *during a first execution* of the program *prior to producing data domain* and testing *during a second execution of the executable computer program* to use data domain including only values that fall within the data domain.

Based on the alternative as to obtain BeanInfo via a introspection APIs in Davidson (see col.. 25 lines 55-58), and the observation of runtime exception (generate error 458 - Fig 4D) in view of the available meta-information (reflection data as beanInfo, e.g. from executing a bean – Table 1, step 422 Fig. 4C) to support verification of bean code or method implementation as shown in Davidson, the concept of using a introspection API during first execution prior to verifying/testing the software via running a second executable version thereof -- testing based on input learned from using first code reflection via introspecting a previous run instance -- has been analyzed in claim 1 using examples of well-known InfoBean methods provided by the Bean Introspector package in Java2SE. Hence, the rationale as to how using collected introspection information from running a first execution instance prior to using such reflection information (e.g. beanInfo) to form data domain, and using the domain input range to validate behaviour of a second instance of code (bean) execution would have been obvious (in view of Davidson combined with Java2SE) herein applies with the same motivation as that set forth in claim 1.

As per claims 20-21, Davidson discloses a graphical user interface communicative with the domain configuration manager for receiving the domain configuration information and transferring (Fig. 1; col. 18-40 -col.28; Error Message – Fig 4B) the domain configuration information to the domain configuration manager; a GUI for receiving user input related to the domain configuration information (e.g. user and application-generated 'events' lines 15-40 -col. 10).

As per claim 22, Davidson discloses domain configuration manager for reading the reflection of the computer program to identify the data structure element for its domain configuration (e.g. Fig. 3B, 3C; Fig. 4B).

As per claim 23, Davidson discloses wherein the data structure element is a data type and the domain configuration manager is operable for producing the data domain for the data type according to an explicit expression indicative of the data domain of the data type (refer to rationale of claims 13-14).

As per claim 24, Davidson discloses wherein the explicit expression comprises methods and functions defined within the computer program (e.g. col. 17-18) and exposed to the domain configuration manager via the reflection of the computer program (Table 1, col. 17-18).

As per claim 25 Davidson discloses wherein the data structure element is a method and the domain configuration manager is operable for producing the data domain as a set of tuples of parameters of the method according to an explicit expression of the domain configuration information (refer to claim 10).

As per claim 26 Davidson discloses wherein the data structure element is a field or a parameter of a declared type and the data configuration manager is operable for producing the data domain according to an explicit expression whose result is an enumeration of values of the declared type (refer to claim 11).

As per claim 27 Davidson discloses wherein the data structure element is a data type with sub-types and the data configuration manager is operable for producing the data domain for the data type through inheritance as a union of data domains of its selected sub-types (refer to claim 13).

As per claim 28 Davidson discloses wherein the data structure element is a data type and the data configuration manager is operable for producing the data domain for the data type by applying a domain generation technique to one or more fields of the data type (refer to claim 15).

As per claim 29, Davidson discloses wherein the domain generation technique is a Cartesian product (e.g. *mapper 122, 124* – Fig. 1; *map 418* – Fig. 4b; *Match 424*, Fig. 4C; *BeanInfo Mapper* - Fig. 5 - Note: a Cartesian or Cross product between 2 sets A and B is defined as the set of all pairs {*a*, *b*} such that *a* is an element of the set A and *b* is an element of the set B; i.e. mapping an element of A with a corresponding element of B) of the selected fields of the data type and the domain configuration manager is further operable for applying a constraint specified in the domain configuration information to the Cartesian product for producing the data domain for the data type (refer to claim 16 for filtering constraint using predicate).

As per claim 30 Davidson discloses wherein the data structure element is a field or a parameter of a declared type and the domain configuration manager is operable for producing the data domain for the field or the parameter as the data domain of their respective declared type through inheritance (refer to claim 14).

As per claim 31, Davidson discloses wherein the data structure element is a method (re claim 24-25) and the domain configuration manager is operable for producing the data domain for the method by applying a domain generation technique to the parameters of the method (re claim 15).

As per claim 32, Davidson discloses wherein the domain configuration technique is a Cartesian product (re claim 29) of the data domains of the parameters (re claims 26 and 30) of the method and the data configuration manager is further operable for applying a constraint (refer to claim 16) for filtering constraint using predicate) to the result of the Cartesian product for producing the data domain for the data type.

As per claim 33, Davidson discloses a computer-based system for producing data domains of data structure elements of an executable computer program to be used to target efficient testing of the program when executed, the executable computer program having been compiled into executable form, the system comprising:

a computer apparatus (Fig. 1);

means for receiving, on the computer apparatus, domain configuration information corresponding to the data structure elements (refer to claim 1);

means for obtaining reflection data (refer to claim 1);

means for processing, on the computer apparatus, the domain configuration information and reflection to produce and output the data domains (refer to claim 19) corresponding to the data structure elements, the data domains representing a limited set of data values to be used as input for testing the computer program (refer to claim 19); and

means for limiting testing to use only values for the data structure element that fall within the data domain (refer to claim 19).

Davidson does not explicitly disclose means for *obtaining*, on the computer apparatus, a *reflection of the executable computer program during a first execution* of the program *prior to producing data domains* with set of data values used as input for testing or for limiting testing (i.e. use value that fall within the data domain) *during a second execution of the computer program*.

But the sequence as to obtain reflection information from a *first execution*, producing range of values in domain based on in part on said reflection data, then effectuating testing of a executable version (*second execution*) of the target code with verification that data values fall within such domain to attest on correct behaviour of the target code has been analyzed as obvious based on the rationale of claims 1, 19, as set forth above.

Response to Arguments

7. Applicant's arguments filed 5/27/09 have been fully considered but they are moot in view of the new grounds of rejection necessitated by the Amendments.

USC § 112 Rejection:

(A) Applicants have submitted that Microsoft ".NET" or Sun Microsystems features that allow executing program to introspect upon itself as shown at page 10, 11, 13 of the Disclosure proves that 'during first execution' is supported (Appl. Rmrks pg. 9-10). The argument has been considered and the grounds of rejection has not been modified, so that the 'during first execution' is not understood in the sense of collecting dynamic reflection data but rather in the context of "reading" such reflection data <u>in conjunction with</u> receiving "domain configuration information", not of which being associated with 'during' execution of any code. The argument is not sufficient to overcome the interpretation and the rejection.

USC § 103 Rejection:

(B) Applicants have submitted that Davidson does not teach 'reading a reflection … during a first execution' (Appl. Rmrks, pg. 11). The interpretation of such 'reading' has been based on the USC 112 Rejection, which is outstanding. The claimed 'reading' has been addressed accordingly, and for claim 1, as amended, this 'during first execution' will not be given weight. Davidson teaches beanInfo as an example of reflection information being read or collected.

(C) Applicants have submitted that Davidson does not actually teach executing a second execution, but rather relates to code implementation, not executable being performed (Appl. Rmrks pg. 11,bottom, pg. 12 top). The target testing being construed as executing a compiled code has been rendered obvious; and regarding the rationale, Applicants are not seen as being able to provide counter-arguments as to how as set forth in the action, the combination set forth in the current 103 rejection would be flawed; that is, the argument mostly moot in view of the newly rearranged rationale of obviousness.

(D) Applicants have submitted that Figure 4D in Davidson relates to a ongoing process of creating program code, clearly prior to (and not done during) execution of the program (Appl. Rmrks pg. 13). The rationale has set forth suggestion, facts and analogous references and their common endeavors as to obtain introspection data based on requirement at bean runtime. The prongs of the 103 rationale are not solely founded on the fact that via the error depicted in Figure 4, Davidson anticipates 'during second execution'. The argument is rather moot because the rationale of rejection has been adjusted based on the latest interpretation of the claim in view of the USC 112 Rejection.

(E) For claims 19 and 33, the grounds of rejection have changed in view of the amendements; the argument no longer commensurate with the previously submitted claim language and/or the new grounds. The arguments (Appl. Rmrks pg. 13-14) become largely moot.

In all, the claims stand rejected as set forth in the Office action.

Conclusion

8. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (571) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Lewis Bullock can be reached on (571)272-3759.

The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence - please consult Examiner before using) or 571-273-8300 (for official correspondence) or redirected to customer service at 571-272-3609.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Tuan A Vu/

Primary Examiner, Art Unit 2193

June 07, 2009